Partie 1

Exercices

1.1 Questions à reponses courtes

a. Quels sont les 2 types numeriques en Python ?

b. Quelle opération donne le quotient de 24 par 4 ?

c. Que donne l'expression : `26 % 4` ?

d. Que donne l'expression : `28 / 7` ?

e. Ecrire la valeur écrite en python $6.67e-11$ (mettre en langage mathématique)

f. Traduire en Python le calcul : $(1, 2.10^{-3}) * *2$

g. Que donne l'expression "'Sup" * 3 ?

1.2 Variables et types

a. Que donne l'expression : `202 + 4`

b. Que donne l'expression : `"202" + str(4)`

c. Que donne l'expression `int(20.2) + 4`

d. Que donne l'expresion suivante : `True and not False` ?

1.3 Variables et opérations

```
1 age = 0
2 annee = 2001
3 age = age + 17
4 annee = annee + age
```

a. Que vaut année à la fin du script ?

b. Que vaut age à la fin du script ?

1.4 nombres de parts

On programme une fonction qui calcule le nombres de parts pour découper les gâteaux en fonction du nombre de personnes :

nom : `nombre_de_parts`

parametress : a : int, nombre de gateaux ; b : int, nombre personnes

valeur de retour : b // a

Voici le script utilisé :

```
# definition de la fonction
def nombre_parts(a,b):
  return b // a

# appel de la fonction
nombre_parts(2,10)
# retourne 5
```

Il faut donc découper chaque gateau en 5 parts.

1.4.1 Appel de la fonction

- Que retourne la fonction dans les cas suivants ?

```
> nombre_parts(2,11)
> nombre_parts(3,32)
```

- Combien de personnes n'auront pas de part de gâteau ?

1.4.2 Amélioration de la fonction

Modifier le script de la fonction afin de proposer une solution qui va permettre de servir tout le monde. Envisager 2 cas :

- Soit le nombre de convives est multiple du nombre de gâteaux : retourner `b // a`
- Soit le nombre de convives n'est pas multiple du nombre de gâteaux : prendre le nombre de convives juste supérieur, multiple du nombre de gâteaux.

1.5 Variables locales et variables globales

1.5.1 Portée d'une variable

Dans une fonction, les variables que l'on créé, et les paramètres utilisés sont des variables locales. Leur *portée* se limite à la seule fonction :

```
> def f(a,b):
        c = 10
        return a * b * c

> f(2,6)
120

> c
NameError                                 Traceback (most recent call
   last)
/var/folders/55/19fzmm8d17ng50cg61f_4chh0000gn/T/ipykernel_62924
   /3235490055.py in <module>
----> 1 c

NameError: name 'c' is not defined
```

Question a : Repérer dans le programme :

- La déclaration de la fonction
- L'appel de fonction
- la consultation de la valeur de la variable, depuis le `main`

Question b : Quel problème est survenu ? Pourquoi

1.5.2 Variable globale

On peut utiliser dans une fonction des variables définies dans le `main`, qui sont alors des variables *globales* :

```
> c = 20
> def f(a,b):
        return a * b * c

> f(2,6)
240
> c
20
```

Question c : Pourquoi le programme ne retourne t'il pas d'erreur lors de l'execution de la fonction ?

1.5.3 Priorité local/global

Exemple de script :

```
a = 20

def f(x):
    a = 10
    return a * x
```

Quelle valeur sera retournée avec l'instruction : `f(1)` ?

Partie 2

COURS - variables et objets en python

2.1 Objets de types primitifs et types construits

Objet et noms

Il faut distinguer en programmation les variables et les valeurs (ou objet). *Une variable est un nom donné à une valeur.* Cette valeur a un type. En python, celui-ci est imposé lors de l'affectation d'une valeur (objet) à la variable créée. (typage dynamique)

Un *objet désigne une entité stockée en mémoire.*

Un objet est toujours d'un certain *type.*

- Les types int, float, bool, list, str sont les types d'objet *primitifs*. Par exemple, un nombre entier est représenté par un objet de la classe int.
- Mais il existe aussi des types construits (tuple, list, dict, function...).

Lorsqu'on créé un nom (c.a.d. une variable), on ajoute dans la *table de noms de l'espace de nom courant* ce nom associé à la référence de l'objet auquel ce nom permet d'accéder.

On peut consulter l'adresse mémoire associé à la valeur grâce à la fonction `id`

```
> a = 10
> print(a)
10
> print(id(a))
1611691328
```

2.2 Variables et espace de nom

Une variable est un nom donné à une valeur, stockée en mémoire. En python, une variable est créée lorsque l'on réalise une affectation. Le nom pointe alors vers une adresse mémoire, dans la RAM :

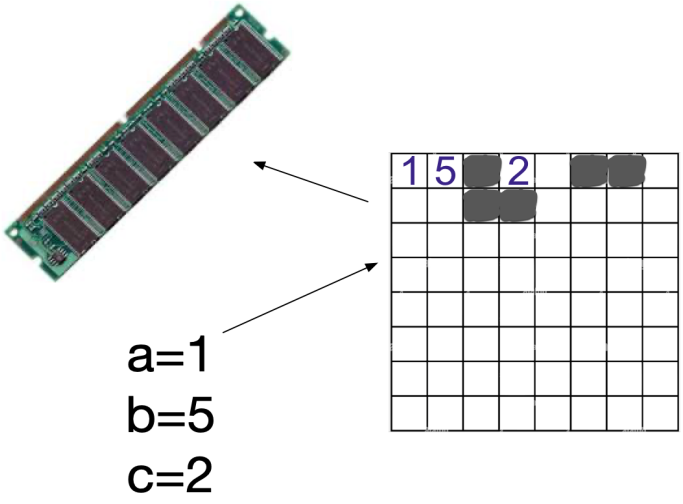

FIGURE 1 – La mémoire : une énorme grille de valeurs et objets

Cette représentation est toutefois incomplète : Au niveau machine, les informations associées au nom de la variable contiennent aussi le *type*. Au niveau logiciel (interpreteur python), les noms de variables sont stockés dans un espace virtuel du programme python : la *table des noms*.

La figure suivante montre l'évolution de la table des noms et des objets lors des trois affectations. L'adresse est l'emplacement en mémoire de l'objet.

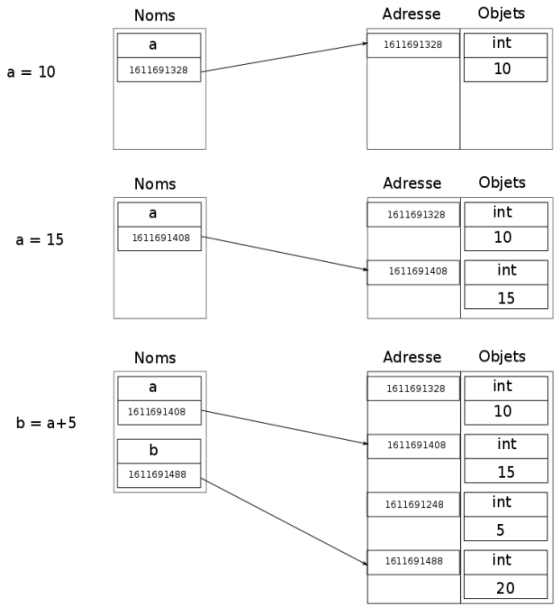

FIGURE 2 – valeur, adresse et affectation

Lorsque l'on fait appel à une variable, le programme consulte l'espace de noms et trouve la valeur en mémoire, puis retourne cette valeur.

2.3 mutable et non mutable

mutable et non mutable

Les types en Python se décompose en deux groupes :

- les types représentant des éléments non-mutable
- les types représentant des éléments mutable

Si une variable est mutable alors on peut modifier son contenu sans créer une nouvelle valeur. Ce sont les *types primitifs* en python, auquels on ajoute les *tuple* et le *type de rien*, `None`. Chaque fois que l'on modifie une variable par une nouvelle affectation, cela modifie son `id`... et créé une nouvelle variable avec le même nom.

Pour les objets non mutables, c'est différents. Ce sont les types *construits*, comme les list, dict, fonctions, classes, objets (issus de classes)...

Partie 3

Fonctions

3.1 Definitions

cours sur allophysique.com

Exercices

Une fonction permet d'encapsuler un bloc d'instructions et de lui donnéer un nom :

```
def ma_fonction():
    # bloc de code ligne 1
    # bloc de code ligne 2
```

La fonction est déclarée après le mot-clé def.

Choisir un nom qui n'appartient pas à Python (aucune autre fonction porte ce nom dans le langage).

Le nom est suivi de parenthèses () et deux points :

Le bloc de code de la fonction est *indenté* sous la ligne def ma_fonction

Appel de fonction : On peut ensuite exécuter ce bloc en utilsant ce nom. On dit qu'on appelle cette fonction en écrivant son nom, suivi de parenthèses ().

```
ma_fonction()
```

Valeur de retour : Une fonction retourne en général une valeur. Après le mot-clé `return`

Une fonction peut posséder des *paramètres*.

Lors de l'appel de la fonction, on place des arguments dans l'ordre des paramètres.

```
# definition de la fonction
def nombre_parts(a,b):
  return b // a

# appel de la fonction
nombre_parts(2,10)
# retourne 5
```

Il y a alors une affectation : paramètre = argument

3.2 Représentation en mémoire

- Lorsqu'on fait appel à une fonction, les instructions de cette fonction opèrent dans un espace de noms propre à la fonction, c'est-à-dire que les noms créés dans la fonction sont créés dans une table particulière appelée espace de noms de la fonction. Ces noms internes à la fonction sont appelées variables locales. Cette table est détruite lorsque la fonction a terminé son exécution.

Ainsi, les noms créés à l'intérieur d'une fonction ont une portée limitée à cette fonction. En revanche, l'espace des noms (c.a.d. la table des noms) utilisée par le code qui appelle la fonction est bien accessible à l'intérieur de la fonction et on parle alors de variables globales.

Exemple :

```
x=10
def f(a):
    b = 7
    return a * b

f(x)
```

pythontutor

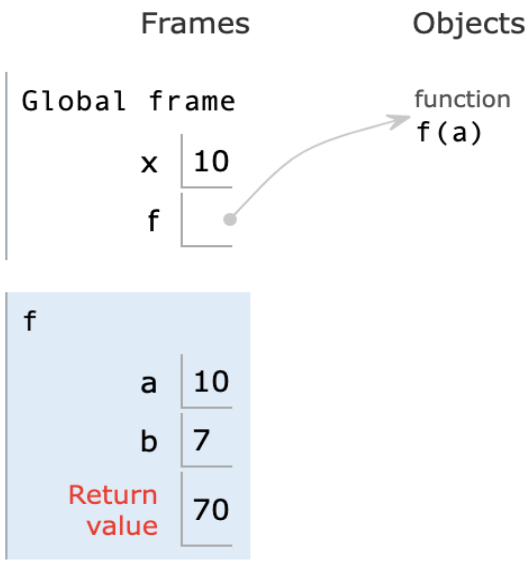

FIGURE 3 – valeur, adresse et affectation

Exercice : Résumer les opérations réalisées sur la mémoire lors de l'appel de la fonction `f(x)`

Partie 4

Portfolio TP1 Python avancé : operations

- Quels sont les types primitifs vus dans cette leçon ? Faire un tableau avec le symbole de tous les opérateurs vus pour chacun de ces types.
- Que donne l'instruction : `type(123e3)`
- Quelle valeur maximale peut prendre un entier signé codé sur 32 bits ?
- Pour les entiers, donner un exemple d'utilisation de l'opérateur `//` et de l'opérateur `%`
- Que donne `x%2` si `x` est pair ? Si `x` est impair ?
- Pour les chaines de caractères, qu'est-ce qu'une concaténation ?
- Donner un exemple d'utilisation du mot clé `in`
- Donner un exemple de comparaison d'ordre lexicographique entre chaines de caractères.
- Soit l'opération logique ci-contre :

```
S and not P
```

`S` peut prendre l'une des 2 valeurs logiques `True` ou `False`. De même que `P`.

Evaluer cette expression pour chaque combinaison de valeurs pour `S` et `P`. Par exemple, pour `S = True` et `P = True`, cette expression devient `True and not True`, qui est alors evaluée à `False`.

Résumer dans un tableau.

- Donner la valeur de S et de P pour représenter l'information suivante :

le temps est ensoleillée (S) et il n'y a pas de pluie (P)

- Donner alors la valeur retournée par l'opération logique `S and not P`

Partie 5

Portfolio TP2 Variables - avancé

- Comment se nomment *en python* les 4 types primitifs que l'on a vus lors de ces premieres séances ?
- Le changement de type entre variables se fait grace aux fonctions `str`, 'float', `int`, et `bool`
  - Comment transformer la chaine "12" en une valeur entière égale à 12 ? "12" => 12
  - Comment réaliser l'opération inverse ? 12 => "12"
  - Comment transformer la chaine "12" en un nombre flottant ? "12" => 12.0
  - Comment transformer l'information 1 en un booléen `True` ?
  - Comment réaliser l'opération inverse ?
- Qu'est-ce qu'une affectation multiple, en une seule ligne d'instruction ?
- Comment échange t-on la valeur de 2 variables `a` et `b` ?
- Pourquoi l'instruction : `print("aujourd'hui j'ai "+ 18 +"ans")` ne fonctionne t-elle pas ? Corriger cette expression (donner 2 moyens).
- Donner un exemple d'utilisation d'une expression formatée pour écrire le résultat du calcul de la force de gravitation $F = G \times m_1 \times m_2/d^2$, à partir des différentes variables.
- Quel sera le type associé à F si l'on réalise le calcul ?